ROSALES, Angel Abegail B.

BSIT 3-5

Activity 5: Using Built-in Functions and Control Structures

Instructions

- Explore R's built-in functions and control structures. Provide R code and a brief explanation for each task. Test your solutions in an R environment.

1. Given a vector of names c("alice", "BOB", "Charlie", "DIANA"), write code to:
   Convert all names to title case (e.g., "Alice", "Bob").
   Add a prefix "ID_" to each name, resulting in names like "ID_Alice".

```
> # Original names
> names <- c("alice", "BOB", "Charlie", "DIANA")
>
> # Convert to title case and add "ID_" prefix
> title_case_names <- tolower(names)  # Convert all names to lowercase
> title_case_names <- tools::toTitleCase(title_case_names)  # Convert to title case
>
> # Add the prefix "ID_"
> prefixed_names <- paste("ID_", title_case_names, sep = "")
>
> # Display the result
> print(prefixed_names)
[1] "ID_Alice"   "ID_Bob"     "ID_Charlie" "ID_Diana"
>
```

   The `tolower()` function changes all names to lowercase. `tools::toTitleCase()` makes the first letter of each word uppercase. `paste()` adds the "ID_" prefix to each name.

2. Write an R function that takes a numeric vector as input and returns a list with the following: Mean, Median, Standard Deviation

```
> # Function to calculate mean, median, and standard deviation
> calculate_stats <- function(num) {
+     result <- list(
+         Mean = mean(num),
+         Median = median(num),
+         StdDev = sd(num)
+     )
+     return(result)
+ }
>
> # Test the function with a random vector
> random_vector <- rnorm(20)  # Generate 20 random numbers
> stats <- calculate_stats(random_vector)
> print(stats)
$Mean
[1] 0.1211962

$Median
[1] 0.1927607

$StdDev
[1] 0.911527
```

   The functions mean(), median(), and sd() are used to calculate the mean, median, and standard deviation, respectively.

3. Test this function with a random vector of 20 values generated using rnorm(). Create a script that takes a numeric vector and classifies each number into:
"Positive" if greater than 0
"Negative" if less than 0
"Zero" otherwise Ensure the script handles edge cases (e.g., empty vector).

```
> # Function to classify numbers
> classify_numbers <- function(num) {
+     if(length(num) == 0) {
+         return("Empty vector")
+     }
+
+     classification <- sapply(num, function(x) {
+         if(x > 0) {
+             return("Positive")
+         } else if(x < 0) {
+             return("Negative")
+         } else {
+             return("Zero")
+         }
+     })
+
+     return(classification)
+ }
>
> # Test with a random vector
> test_vector <- rnorm(20)
> classification_result <- classify_numbers(test_vector)
> print(classification_result)
 [1] "Negative" "Negative" "Negative" "Negative" "Negative" "Negative" "Positive"
 [8] "Negative" "Positive" "Positive" "Positive" "Positive" "Positive" "Negative"
[15] "Positive" "Negative" "Negative" "Positive" "Positive" "Negative"
>
```

The `sapply()` function applies a function to every element in the vector. The function labels each number as "Positive", "Negative", or "Zero" depending on its value. It also handles the edge case of an empty vector by checking its length.

4. Given a vector of scores c(70, 85, 90, 65, 95, 88), write code to calculate the average score only for scores above 80.

```
> # Vector of scores
> scores <- c(70, 85, 90, 65, 95, 88)
>
> # Filter scores greater than 80
> above_80_scores <- scores[scores > 80]
>
> # Calculate the average
> average_score <- mean(above_80_scores)
> print(average_score)
[1] 89.5
>
```

The condition scores > 80 filters out scores greater than 80 and then, mean() calculates the average of the filtered scores.

5. Write a recursive R function to compute the nth Fibonacci number. Demonstrate the function with inputs n = 5 and n = 10.

```
> # Recursive function to compute nth Fibonacci number
> fibonacci <- function(n) {
+     if (n <= 1) {
+         return(n)  # Base case: return n if n is 0 or 1
+     } else {
+         return(fibonacci(n - 1) + fibonacci(n - 2))  # Recursive call
+     }
+ }
>
> # Demonstration with n = 5 and n = 10
> cat("Fibonacci(5):", fibonacci(5), "\n")  # Output for n = 5
Fibonacci(5): 5
> cat("Fibonacci(10):", fibonacci(10), "\n")  # Output for n = 10
Fibonacci(10): 55
>
```

The function fibonacci(n) computes the Fibonacci number for a given n. If n is 0 or 1, it returns n as the base case. Otherwise, it recursively calls fibonacci(n - 1) and fibonacci(n - 2) to sum the previous two Fibonacci numbers.